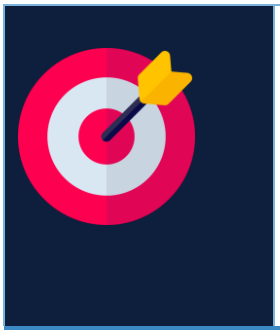
Desarrollo Seguro en Python

Lección 3: OWASP TOP 10 Controles Proactivos

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

OWASP TOP 10 CONTROLES PROACTIVOS



Objetivos

- Conocer los principales controles proactivos.
- Conocer herramientas y métodos para poder aplicarlos.
- Ver ejemplos de aplicación

1. INTRODUCCIÓN

Para lograr un software seguro, los desarrolladores deben contar con el respaldo y la ayuda de la organización para la que escriben código, cosa que, a veces, no sucede. A media que los desarrolladores de software crean el código que compone una aplicación, se necesita adoptar y practicar una amplia variedad de técnicas de codificación segura. Todas las capas que componen una aplicación, tales como la interfaz de usuario, la lógica de negocio, el controlador o la base de datos, deben desarrollarse teniendo en la seguridad en mente. La mayoría de los desarrolladores no han aprendido sobre codificación segura o la criptografía en su etapa de formación y, en ocasiones, tampoco se lo han exigido.

Los lenguajes y frameworks que los desarrolladores usan para el desarrollo de las aplicaciones normalmente carecen de controles centrales críticos o son inseguros por defecto de alguna manera. No es normal que las organizaciones den a los desarrolladores requisitos prescriptivos que los guíen por el camino del software seguro e, incluso, cuando lo hacen, puede haber errores de seguridad inherentes a los requisitos y diseños.

Por ello, se ha de intentar se proactivo desde los inicios del diseño del proyecto que se realice, teniendo en cuenta, en principio, la seguridad.

El TOP 10 de controles proactivos de OWASP es una lista de técnicas de seguridad que deben considerarse para cada proyecto de desarrollo software seguro.

Uno de los principales objetivos de esta lista es proporcionar una guía práctica concreta que ayude a los desarrolladores a crear software seguro. Estas técnicas deben aplicarse de forma proactiva en las primeras etapas del desarrollo de software para garantizar la máxima eficacia.

La siguiente lista está ordenada por importancia.

- C1: Definir requisitos de seguridad
- C2: Aprovechar las bibliotecas y los frameworks de seguridad
- C3: Acceso seguro a la base de datos
- C4: Codificar y escapar datos
- C5: Validar todas las entradas
- C6: Implementar la identidad digital
- C7: Aplicar controles de acceso
- C8: Proteja los datos en todas partes
- C9: Implementar el registro y la supervisión de seguridad
- C10: Manejar todos los errores y excepciones

CONTROLES PROACTIVOS

1. DEFINIR LOS REQUISITOS DE SEGURIDAD

Un requisito de seguridad es una declaración de la funcionalidad de seguridad necesaria que garantiza que se satisfaga una de las muchas propiedades de seguridad diferentes del software.

Los requisitos de seguridad se derivan de los estándares de la industria, las leyes aplicables y un historial de vulnerabilidades pasadas. Los requisitos de seguridad definen nuevas funciones o añaden características a las existentes para resolver un problema de seguridad específico o eliminar una vulnerabilidad potencial.

Los requisitos de seguridad proporcionan una base de funcionalidad de seguridad comprobada para una aplicación. En lugar de crear un enfoque de seguridad personalizado para cada aplicación, los requisitos de seguridad estándar permiten a los desarrolladores reutilizar la definición de controles de seguridad y mejores prácticas. Esos mismos requisitos de seguridad examinados proporcionan soluciones para problemas de seguridad que se han producido en el pasado. Existen requisitos para evitar la repetición de fallos de seguridad pasadas.

OWASP ASVS

OWASP Application Security Verification Standard (ASVS) es un catálogo de requisitos de seguridad y criterios de verificación disponibles. OWASP ASVS puede ser una fuente de requisitos de seguridad detallados para los equipos de desarrollo.

Los requisitos de seguridad se clasifican en diferentes grupos según una función de seguridad de orden superior compartida. Por ejemplo, el ASVS contiene categorías como autenticación, control de acceso, manejo / registro de errores y servicios web.

Cada categoría contiene una colección de requisitos que representan las mejores prácticas para esa categoría redactadas como declaraciones verificables.

Aumento de requisitos con historias de usuarios y casos de uso indebido

Presta atención

Los requisitos de ASVS son declaraciones verificables básicas que se pueden ampliar con historias de usuarios y casos de uso indebido.

La ventaja de una historia de usuario o un caso de uso indebido es que vincula la aplicación exactamente con lo que el usuario o atacante hace con el sistema, en lugar de describir lo que el sistema ofrece al usuario.

A continuación, se muestra un ejemplo de aplicación de un requisito de ASVS 3.0.1. De la sección "Requisitos de verificación de autenticación" de ASVS 3.0.1, el requisito 2.19 se centra en las contraseñas predeterminadas.

2.19 Verifique que no haya contraseñas predeterminadas en uso para el marco de la aplicación o cualquier componente utilizado por la aplicación (como "admin / password").

Este requisito contiene una acción para verificar que no existan contraseñas predeterminadas y también incluye la guía de que no deben usarse contraseñas predeterminadas dentro de la aplicación.

Una historia de usuario se centra en la perspectiva del usuario, administrador o atacante del sistema y describe la funcionalidad en función de lo que un usuario quiere que el sistema haga por él. Una historia de usuario toma la forma de "Como usuario, puedo hacer x, y, z".

Como usuario, puedo ingresar mi nombre de usuario y contraseña para acceder a la aplicación.

Como usuario, puedo ingresar una contraseña larga que tenga un máximo de 1023 caracteres.

Cuando la historia se centra en el atacante y sus acciones, se denomina caso de uso indebido.

Como atacante, puedo ingresar un nombre de usuario y contraseña predeterminados para obtener acceso.

Esta historia contiene el mismo mensaje que el requisito tradicional de ASVS, con detalles adicionales del usuario o atacante para ayudar a que el requisito sea más comprobable.

Implementación

El uso exitoso de los requisitos de seguridad implica cuatro pasos. El proceso incluye descubrir / seleccionar, documentar, implementar y luego confirmar la implementación correcta de nuevas características y funcionalidades de seguridad dentro de una aplicación.

Descubrimiento y selección

El proceso comienza con el descubrimiento y la selección de requisitos de seguridad. En esta fase, el desarrollador comprende los requisitos de seguridad de una fuente estándar como ASVS y elige qué requisitos incluir para una versión determinada de una aplicación. El punto de descubrimiento y selección es elegir una cantidad manejable de requisitos de seguridad para esta versión o sprint, y luego continuar iterando para cada sprint, agregando más funciones de seguridad con el tiempo.

Investigación y documentación

Durante la investigación y la documentación, el desarrollador revisa la aplicación existente con el nuevo conjunto de requisitos de seguridad para determinar si la aplicación cumple actualmente con el requisito o si se requiere algún desarrollo. Esta investigación culmina con la documentación de los resultados de la revisión.

Implementación y prueba

Una vez que se determina la necesidad de desarrollo, el desarrollador ahora debe modificar la aplicación de alguna manera para agregar la nueva funcionalidad o eliminar una opción insegura.

En esta fase, el desarrollador primero determina el diseño requerido para abordar el requisito y luego completa los cambios de código para cumplir con el requisito. Se deben crear casos de prueba para confirmar la existencia de la nueva funcionalidad o refutar la existencia de una opción previamente insegura.

Vulnerabilidades evitadas

Los requisitos de seguridad definen la funcionalidad de seguridad de una aplicación. Una mejor seguridad incorporada desde el comienzo del ciclo de vida de las aplicaciones da como resultado la prevención de muchos tipos de vulnerabilidades.

Referencias:

- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

2. APROVECHAR LAS LIBRERÍAS Y LOS FRAMEWORKS DE SEGURIDAD

Descripción

Las librerías de codificación seguras y los frameworks con seguridad integrada ayudan a los desarrolladores de software a protegerse contra los fallos de implementación y diseño relacionados con la seguridad. Un desarrollador que escribe una aplicación desde cero puede no tener el conocimiento, el tiempo o el presupuesto suficiente para implementar o mantener las características de seguridad de manera adecuada. Aprovechar los frameworks de seguridad ayuda a lograr los objetivos de seguridad de manera más eficiente y precisa.

Mejores prácticas de implementación

Al incorporar librerías o frameworks de terceros en tu software, es importante tener en cuenta las siguientes buenas prácticas.

1. Utiliza librerías y frameworks de fuentes confiables que se mantienen activamente y son ampliamente utilizados por muchas aplicaciones.
2. Cree y mantenga un catálogo de inventario de todas las bibliotecas de terceros.
3. Mantenga actualizadas las bibliotecas y los componentes de forma proactiva. Usa herramientas como OWASP Dependency Check y Retire.JS para identificar las dependencias del proyecto y verificar si hay vulnerabilidades conocidas y divulgadas públicamente para todo el código de terceros.
4. Reduzca la superficie de ataque encapsulando las librerías y solo exponga el comportamiento requerido en su software.

Vulnerabilidades evitadas

Los frameworks y librerías seguras pueden ayudar a prevenir un amplio rango de vulnerabilidades en aplicaciones web.

Es fundamental mantener estos frameworks y librerías actualizados como se describe en el uso de componentes con vulnerabilidades conocidas.

Herramientas

OWASP Dependency-Check: Identifica las dependencias del proyecto y comprueba las vulnerabilidades divulgadas públicamente.

Retire.JS escáner para librerías JavaScript.

3. ACCESO SEGURO A LA BASE DE DATOS

Esta sección describe el acceso seguro a todos los almacenes de datos, incluidas las bases de datos relacionales y las bases de datos NoSQL.

Algunas áreas a considerer:

- Consultas seguras
- Configuración segura
- Autenticación segura
- Comunicación segura

Consultas seguras

SQL Injection ocurre cuando la entrada de un usuario que no es de confianza se agrega dinámicamente a una consulta SQL de una manera insegura, a menudo a través de una concatenación básica de cadenas.

SQL Injection es uno de los riesgos de seguridad de aplicaciones más peligrosos. La inyección SQL es fácil de explotar y podría provocar el robo, borrado o modificación de toda la base de datos. La aplicación incluso se puede utilizar para ejecutar comandos peligrosos contra el sistema operativo que aloja su base de datos, lo que le da a un atacante un punto de apoyo en su red.

Para mitigar el **SQL Injection** se debe evitar que la entrada que no sea de confianza se interprete como parte de un comando SQL. La mejor forma de hacer esto es con la técnica de programación conocida como "parametrización de consultas". Este defensa debe aplicarse a SQL, OQL, así como a la construcción de procedimientos almacenados.

Puede encontrarse una buena lista de ejemplos de parametrización de consultas en ASP , ColdFusion , C# , Delphi, .NET , Go , Java , Perl , PHP , PL/SQL , PostgreSQL, Python , R , Ruby and Scheme en http://bobby-tables.com y OWASP Cheat Sheet on Query Parameterization

Precaución sobre la parametrización de consultas

Ciertas ubicaciones en una consulta a base de datos no se pueden parametrizar. Estas ubicaciones son diferentes para cada proveedor de base de datos. Asegúrate de realizar una validación de coincidencia exacta o un escape manual con mucho cuidado cuando se enfrente a parámetros de consulta de base de datos que no se puedan vincular a una consulta parametrizada. Además, aunque el uso de consultas parametrizadas tiene en gran medida un impacto positivo en el rendimiento, ciertas consultas parametrizadas en implementaciones de base de datos específicas afectarán negativamente al rendimiento. Asegúrese de probar el rendimiento de las consultas; consultas especialmente complejas con amplias capacidades de búsqueda de texto o cláusulas similares.

Configuración segura

Desafortunadamente, los sistemas de administración de bases de datos no siempre se envían con una configuración segura por defecto. Se debe tener cuidado para garantizar que los controles de seguridad disponibles en el sistema de administración de bases de datos (DBMS) y la plataforma de alojamiento estén habilitados y configurados correctamente. Hay estándares, guías y puntos de referencia disponibles para los DBMS más comunes.

Autenticación segura

Todo acceso a la base de datos debe estar debidamente autenticado. La autenticación en el DBMS debe realizarse de forma segura. La autenticación debe tener lugar solo a través de un canal seguro. Las credenciales deben estar debidamente aseguradas y disponibles para su uso.

Importante

La mayoría de los DBMS admiten una variedad de métodos de comunicación (servicios, API, etc.): seguros (autenticados, cifrados) e inseguros (no autenticados o no cifrados). Es una buena práctica utilizar solo las opciones de comunicaciones seguras según el control ***Proteger Datos en Todas Partes***

Vulnerabilidades evitadas

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

Referencias

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

4. CODIFICAR Y ESCAPAR DATOS

Codificar y escapar son técnicas defensivas destinadas a detener los ataques de inyección. La codificación (comúnmente conocida como “Codificación de salida") implica traducir caracteres especiales a una forma diferente pero equivalente que ya no es peligrosa en el intérprete de destino, por ejemplo, traducir el carácter “<” a la cadena < cuando se escribe en una página HTML.

Escapar implica agregar un carácter especial antes del carácter / cadena para evitar que se malinterprete, por ejemplo, agregar un carácter “\” antes de un carácter “ " ” (comillas dobles) para que se interprete como texto y no como cierre de cadena.

La codificación de salida se aplica mejor justo antes de que el contenido se pase al intérprete de destino. Si esta defensa se realiza demasiado pronto en el procesamiento de una solicitud, la codificación o el escape pueden interferir con el uso del contenido en otras partes del programa. Por ejemplo, si tu HTML escapa contenido antes de almacenar esos datos en la base de datos y la interfaz de usuario escapa automáticamente estos datos por segunda vez, el contenido no se mostrará correctamente debido a que tiene un doble escape.

Presta atención

La codificación de salida se aplica mejor justo antes de que el contenido se pase al intérprete de destino.

Codificación de salida contextual

La codificación de salida contextual es una técnica de programación de seguridad crucial necesaria para detener XSS.

Esta defensa se realiza en la salida, cuando está creando una interfaz de usuario, en el último momento antes de que los datos que no son de confianza se agreguen dinámicamente al HTML.

El tipo de codificación dependerá de la ubicación (o contexto) en el documento donde se muestran o almacenan los datos. Los diferentes tipos de codificación que se utilizarían para crear interfaces de usuario seguras incluyen codificación de entidad HTML, codificación de atributo HTML, codificación de JavaScript y codificación de URL.

Ejemplos de codificación de Java

Para ver ejemplos del codificador Java OWASP que proporciona codificación de salida contextual, ver: OWASP Java Encoder Project Examples.

Ejemplos de codificación .NET

A partir de .NET 4.5, la biblioteca Anti-Cross Site Scripting es parte del framework, pero no está habilitada por defecto. Puede especificar usar AntiXssEncoder de esta biblioteca como el codificador predeterminado para toda su aplicación utilizando el web.conf settings.

Cuando se aplica, es importante codificar contextualmente su salida, lo que significa usar la función correcta de la biblioteca AntiXSSEncoder para la ubicación adecuada de los datos en el documento.

Ejemplos de codificación Python

Django https://docs.djangoproject.com/es/3.0/_modules/django/utils/html/

Otros tipos de codificación y defensa de inyección

Codificación / escapado se puede utilizar para neutralizar el contenido frente a otras formas de inyección.

Por ejemplo, es posible neutralizar ciertos metacaracteres especiales al agregar entrada a un comando del sistema operativo. Esto se denomina "escape de comandos del sistema operativo", "escape de shell" o similar. Esta defensa se puede utilizar para detener las vulnerabilidades de "inyección de comandos".

Hay otras formas de escape que se pueden usar para detener la inyección, como el escape de atributos XML que detiene varias formas de inyección de ruta XML y XML, así como el escape de nombre distinguido LDAP que se puede usar para detener varias formas de inyección LDAP.

Codificación y canonicalización de caracteres

La codificación Unicode es un método para almacenar caracteres con varios bytes.

Dondequiera que se permita la entrada de datos, los datos se pueden ingresar usando Unicode para disfrazar el código malicioso y permitir una variedad de ataques. RFC 2279 hace referencia a muchas formas en que se puede codificar el texto.

La canonicalización es un método en el que los sistemas convierten los datos en una forma simple o estándar. Las aplicaciones web suelen utilizar la canonicalización de caracteres para garantizar que todo el contenido sea del mismo tipo de caracteres cuando se almacena o muestra.

Para estar seguro contra los ataques relacionados con la canonicalización, una aplicación debe ser segura cuando se ingresan Unicode con formato incorrecto y otras representaciones de caracteres con formato incorrecto.

Vulnerabilidades prevenidas

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client Side Injection

Referencias

- XSS - General information
- OWASP Cheat Sheet: XSS Prevention - Stopping XSS in your web application.
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

Herramientas

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper - examples of contextual encoding

5. VALIDAR TODAS LAS ENTRADAS

La validación de entrada es una técnica de programación que garantiza que solo los datos formateados correctamente puedan entrar a un componente del sistema de software.

Validez sintáctica y semántica

Una aplicación debe verificar que los datos sean tanto sintácticamente como semánticamente válidos (en ese orden) antes de usarlos de cualquier manera (incluida la visualización al usuario).

La **validez de sintaxis** significa que los datos están en la forma esperada. Por ejemplo, una aplicación puede permitir que un usuario seleccione un "ID de cuenta" de cuatro dígitos para realizar algún tipo de operación.

La aplicación debe asumir que el usuario está ingresando una carga de inyección de SQL y debe verificar que los datos ingresados por el usuario tengan exactamente cuatro dígitos de longitud y consistan solo en números (además de utilizar la parametrización de consultas adecuada).

La **validez semántica** incluye solo aceptar entradas que estén dentro de un rango aceptable para la funcionalidad y el contexto de la aplicación. Por ejemplo, una fecha de inicio debe ser anterior a una fecha de finalización al elegir rangos de fechas.

Lista blanca contra lista negra

Hay dos enfoques generales para realizar la validación de la sintaxis de entrada, comúnmente conocidos como listas negras y listas blancas.

- **La validación de listas negras o listas negras** intenta verificar que los datos proporcionados no contengan contenido "conocido incorrecto". Por ejemplo, una aplicación web puede bloquear la entrada que contiene el texto exacto <SCRIPT> para ayudar a prevenir XSS. Sin embargo, esta defensa podría eludirse con una etiqueta de secuencia de comandos en minúscula o una etiqueta de secuencia de comandos de mayúsculas y minúsculas.
- **La validación de la lista blanca o lista blanca** intenta comprobar que un dato dado coincide con un conjunto de reglas de "bien conocido". Por ejemplo, una regla de validación de lista blanca para un estado de EE. UU. Sería un código de 2 letras que es solo uno de los estados válidos de EE. UU.

Al crear software seguro, la inclusión en listas blancas es el enfoque mínimo recomendado. Las listas negras son propensas a errores y pueden evitarse con varias técnicas de evasión. Pueden ser peligrosas cuando se depende de ellas por sí mismas.

Aunque las listas negras a menudo se pueden eludir, también pueden ser útiles para ayudar a detectar ataques obvios. Por lo tanto, si bien las listas blancas ayudan a limitar la superficie de ataque al garantizar que los datos tengan la validez sintáctica y semántica correcta, las listas negras ayudan a detectar y detener ataques potencialmente obvios.

Validación del lado del cliente y del lado del servidor

La validación de entrada siempre debe realizarse en el lado del servidor por motivos de seguridad. Si bien la validación del lado del cliente puede ser útil tanto para fines funcionales como de seguridad, a menudo se puede omitir fácilmente. Esto hace que la validación del lado del servidor sea aún más fundamental para la seguridad.

Por ejemplo, la validación de JavaScript puede alertar al usuario de que un campo en particular debe constar de números, pero la aplicación del lado del servidor debe validar que los datos enviados solo constan de números en el rango numérico apropiado para esa característica.

Expresiones regulares

Las expresiones regulares ofrecen una forma de verificar si los datos coinciden con un patrón específico. Comencemos con un ejemplo básico.

La siguiente expresión regular se utiliza para definir una regla de lista blanca para validar nombres de usuario.

```
^[a-z0-9_]{3,16}$
```

Esta expresión regular solo permite letras minúsculas, números y el carácter de subrayado. El nombre de usuario también está restringido a una longitud de 3 y 16 caracteres.

Precaución: potencial de denegación de servicio

Se debe tener cuidado al crear expresiones regulares. Las expresiones mal diseñadas pueden resultar en posibles condiciones de denegación de servicio (también conocidas como ReDoS). Varias herramientas pueden probar para verificar que las expresiones regulares no sean vulnerables a ReDoS.

Precaución: complejidad

Las expresiones regulares son solo una forma de lograr la validación. Las expresiones regulares pueden resultar difíciles de mantener o comprender para algunos desarrolladores. Otras alternativas de validación implican escribir métodos de validación mediante programación que pueden ser más fáciles de mantener para algunos desarrolladores.

Límites de la validación de entrada

La validación de entrada no siempre hace que los datos sean “seguros” ya que ciertas formas de entrada compleja pueden ser “válidas” pero aún peligrosas. Por ejemplo, una dirección de correo electrónico válida puede contener un ataque de inyección SQL o una URL válida puede contener un ataque Cross Site Scripting.

Siempre se deben aplicar defensas adicionales además de la validación de entrada a datos como la parametrización de consultas o el escape.

Desafíos de la validación de datos serializados

Algunas formas de entrada son tan complejas que la validación solo puede proteger mínimamente la aplicación. Por ejemplo, es peligroso deserializar datos que no son de confianza o datos que pueden ser manipulados por un atacante. El único patrón de arquitectura seguro es no aceptar objetos serializados de fuentes no confiables o deserializar solo en una capacidad limitada solo para tipos de datos simples. Debe evitar procesar formatos de datos serializados y usar formatos más fáciles de defender como JSON cuando sea posible.

Si eso no es posible, entonces se consideran una serie de defensas de validación al procesar datos serializados.

- Implemente controles de integridad o encriptación de los objetos serializados para evitar la creación de objetos hostiles o la manipulación de datos.
- Aplique restricciones de tipo estrictas durante la deserialización antes de la creación del objeto; normalmente, el código espera un conjunto definible de clases. Se han demostrado desvíos de esta técnica.
- Aísle el código que se deserializa, de modo que se ejecute en entornos con muy pocos privilegios, como contenedores temporales.
- Registre las excepciones y fallos de deserialización de seguridad, como cuando el tipo entrante no es el tipo esperado, o la deserialización genera excepciones.
- Restrinja o supervise la conectividad de red entrante y saliente de contenedores o servidores que deserializan.
- Supervise la deserialización, alertando si un usuario deserializa constantemente.

Entrada de usuario inesperada (asignación masiva)

Algunos frameworks admiten el enlace automático de los parámetros de las solicitudes HTTP a los objetos del lado del servidor que utiliza la aplicación. Esta función de enlace automático puede permitir a un atacante actualizar objetos del lado del servidor que no estaban destinados a ser modificados. El atacante posiblemente pueda modificar su nivel de control de acceso o eludir la lógica de negocio prevista de la aplicación con esta función.

Este ataque tiene varios nombres que incluyen: asignación masiva, enlace automático e inyección de objetos.

Como ejemplo simple, si el objeto de usuario tiene un campo privilegio que especifica el nivel de privilegio del usuario en la aplicación, un usuario malintencionado puede buscar páginas donde se modifiquen los datos del usuario y agregar privilege = admin a los parámetros HTTP enviados. Si el enlace automático está habilitado de forma insegura, el objeto del lado del servidor que representa al usuario se modificará en consecuencia.

Se pueden usar dos métodos para manejar esto:

- Evite vincular la entrada directamente y utilice objetos de transferencia de datos (DTO) en su lugar.
- Habilite el enlace automático, pero configure las reglas de lista blanca para cada página o función para definir qué campos pueden enlazarse automáticamente.

Más ejemplos están disponibles en OWASP Mass Assignment Cheat Sheet.

Validación y limpieza de HTML

Considere una aplicación que necesita aceptar HTML de los usuarios (a través de un editor WYSIWYG que representa el contenido como HTML o funciones que aceptan HTML directamente en la entrada). En esta situación, la validación o el escape no ayudarán.

- Las expresiones regulares no son lo suficientemente expresivas para comprender la complejidad de HTML5.

- Codificar o escapar de HTML no ayudará ya que hará que el HTML no se procese correctamente.

Por lo tanto, necesita una biblioteca que pueda analizar y limpiar texto con formato HTML. Consulte la XSS Prevention Cheat Sheet on HTML Sanitization para más información sobre la limpieza de HTML.

Funcionalidad de validación en librerías y Frameworks

Todos los lenguajes y la mayoría de los frameworks proporcionan librerías de validación o funciones las cuales deben ser aprovechadas para validar datos.

Las librerías de validación generales cubren tipos de datos comunes, requisitos de longitud, rangos de números enteros, verificaciones "es nulo" y más. Muchas librerías y frameworks de validación te permiten definir tu propia expresión regular o lógica para la validación personalizada de una manera que le permite al programador aprovechar esa funcionalidad en toda su aplicación.

Ejemplos de funcionalidad de validación incluyen las funciones de filtro de PHP o Hibernate Validator para Java. Ejemplos de sanitización de HTML incluyen Ruby on Rails sanitize method, OWASP Java HTML Sanitizer or DOMPurify.

Vulnerabilidades evitadas

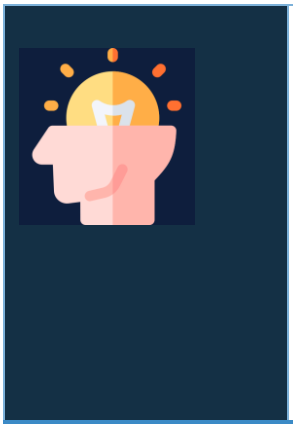
- La validación de entrada reduce la superficie de ataque de las aplicaciones y, a veces, puede dificultar los ataques contra una aplicación.
- La validación de entrada es una técnica que proporciona seguridad a determinadas formas de datos, específicas de determinados ataques y no se puede aplicar de forma fiable como regla de seguridad general.
- La validación de entrada no debe usarse como método principal para prevenir XSS, inyección SQL y otros ataques.

Referencias

- OWASP Cheat Sheet: Input Validation

Herramientas

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP's filter functions



No te olvides...

Validación sintáctica, semántica. Uso de listas blancas y negras. Validación lado cliente y servidor. Uso de expresiones regulares. Validación y limpieza de HTML y funcionalidad de validación en librerías y Frameworks.

6. IMPLEMENTAR LA IDENTIDAD DIGITAL

La identidad digital es la representación única de un usuario (u otro sujeto) mientras realiza una transacción en línea. La autenticación es el proceso de verificar que una persona o entidad es quien dice ser.

La gestión de sesiones es un proceso mediante el cual un servidor mantiene el estado de la autenticación de los usuarios para que el usuario pueda continuar utilizando el sistema sin volver a autenticarse.

La NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management) proporciona una guía sólida sobre la implementación de controles de administración de sesión, autenticación e identidad digital.

A continuación, se presentan algunas recomendaciones para una implementación segura.

Niveles de autenticación

NIST 800-63b describe tres niveles de garantía de autenticación denominados nivel de garantía de autenticación (AAL). El nivel 1 de AAL está reservado para aplicaciones de menor riesgo que no contienen PII u otros datos privados. En el nivel 1 de AAL solo se requiere autenticación de factor único, generalmente mediante el uso de una contraseña.

Nivel 1. Contraseñas.

Las contraseñas son realmente importantes. Necesitamos una política, necesitamos almacenarlos de forma segura, a veces debemos permitir que los usuarios las restablezcan.

Requisitos de contraseña

Las contraseñas deben cumplir como mínimo los siguientes requisitos:

- Tener al menos 8 caracteres de longitud si también se utilizan la autenticación multifactor (MFA) y otros controles. Si MFA no es posible, debe aumentarse a al menos 10 caracteres.
- Todos los caracteres ASCII impresos, así como el carácter de espacio, deben ser aceptables en los secretos memorizados
- Fomentar el uso de contraseñas largas
- Eliminar los requisitos de complejidad, ya que se ha encontrado que tienen una eficacia limitada. En su lugar, se recomienda la adopción de MFA o contraseñas de mayor longitud.
- Asegúrese de que las contraseñas utilizadas no sean contraseñas de uso común que ya se hayan filtrado en un compromiso anterior. Puede optar por bloquear las 1000 o 10000 contraseñas más comunes que cumplen los requisitos de longitud anteriores y se encuentran en listas de contraseñas comprometidas. El siguiente enlace contiene las contraseñas más comunes:

  https://github.com/danielmiessler/SecLists/tree/master/Passwords

Implementar un mecanismo de recuperación de contraseña segura

Es común que una aplicación tenga un mecanismo para que un usuario obtenga acceso a su cuenta en caso de que olvide su contraseña. Un buen flujo de trabajo de diseño para una función de recuperación de contraseña utilizará elementos de autenticación de múltiples factores. Por ejemplo, puede hacer una pregunta de seguridad, algo que saben, y luego enviar un token generado a un dispositivo, algo que poseen.

Implementar almacenamiento seguro de contraseñas

Para proporcionar controles de autenticación sólidos, una aplicación debe almacenar de forma segura las credenciales de usuario. Además, deben existir controles criptográficos de modo que si una credencial (por ejemplo, una contraseña) se ve comprometida, el atacante no tenga acceso inmediato a esta información.

Ejemplo de PHP para almacenamiento de contraseñas

A continuación, se muestra un ejemplo de hash seguro de contraseñas en PHP usando la función password_hash () (disponible desde 5.5.0) que por defecto usa el algoritmo bcrypt. El ejemplo usa un factor de trabajo de 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

Nivel 2: autenticación multifactor

NIST 800-63b AAL nivel 2 está reservado para aplicaciones de mayor riesgo que contienen "PII (Información Personal Identificable) autoafirmada u otra información personal disponible en línea". En el nivel 2 de AAL, se requiere la autenticación multifactor, incluida la OTP (contraseña válida para una autenticación) u otras formas de implementación multifactor.

La autenticación multifactor (MFA) garantiza que los usuarios sean quienes dicen ser al exigirles que se identifiquen con una combinación de:

- Algo que sepa: contraseña o PIN
- Algo de su propiedad: una ficha o un teléfono
- Algo que eres: datos biométricos, como una huella digital

El uso de contraseñas como único factor proporciona una seguridad débil. Las soluciones multifactoriales proporcionan una solución más sólida al requerir que un atacante adquiera más de un elemento para autenticarse con el servicio.

Vale la pena señalar que la biometría, cuando se emplea como factor único de autenticación, no se considera aceptable para la autenticación digital.

Se pueden obtener en línea o tomando una fotografía de alguien con un teléfono con cámara (por ejemplo, imágenes faciales) con o sin su conocimiento, levantadas de objetos que alguien toca (por ejemplo, huellas dactilares latentes) o capturadas con imágenes de alta resolución (por ejemplo, iris patrones).

La biometría debe usarse solo como parte de la autenticación de múltiples factores con un autenticador físico (algo que usted posee). Por ejemplo, acceder a un dispositivo de contraseña de un solo uso (OTP) de múltiples factores que generará una contraseña de un solo uso que el usuario ingresa manualmente para el verificador.

Nivel 3: autenticación basada en criptografía

NIST 800-63b Authentication Assurance Level 3 (AAL3) se requiere cuando el impacto de los sistemas comprometidos podría conducir a daños personales, pérdidas financieras significativas, dañar el interés público o involucrar violaciones civiles o criminales. AAL3 requiere autenticación que se "basa en la prueba de posesión de una clave a través de un protocolo criptográfico". Este tipo de autenticación se utiliza para lograr el mayor nivel de garantía de autenticación. Esto se hace normalmente a través de módulos criptográficos de hardware.

Manejo de sesiones

Una vez que ha tenido lugar la autenticación inicial exitosa del usuario, una aplicación puede optar por rastrear y mantener este estado de autenticación durante un período de tiempo limitado.

Esto permitirá al usuario continuar usando la aplicación sin tener que volver a autenticarse con cada solicitud. El seguimiento de este estado de usuario se denomina Gestión de sesiones.

Generación y caducidad de sesiones

El estado del usuario se rastrea en una sesión. Esta sesión normalmente se almacena en el servidor para la gestión de sesiones tradicional basada en web. A continuación, se le da al usuario un identificador de sesión para que pueda identificar qué sesión del lado del servidor contiene los datos de usuario correctos. El cliente solo necesita mantener este identificador de sesión, que también mantiene los datos confidenciales de la sesión del lado del servidor fuera del cliente.

A continuación, se muestran algunos controles que se deben tener en cuenta al crear o implementar soluciones de administración de sesiones:

- Asegúrese de que la identificación de la sesión sea larga, única y aleatoria.
- La aplicación debe generar una nueva sesión o al menos rotar la identificación de la sesión durante la autenticación y reautenticación.
- La aplicación debe implementar un tiempo de espera inactivo después de un período de inactividad y una vida útil máxima absoluta para cada sesión, después del cual los usuarios deben volver a autenticarse. La duración de los tiempos de espera debe ser inversamente proporcional al valor de los datos protegidos.

Consulte la hoja de referencia de gestión de sesiones para obtener más detalles. La Sección 3 de ASVS cubre los requisitos adicionales de gestión de sesiones.

Cookies del navegador

Las cookies del navegador son un método común para que las aplicaciones web almacenen identificadores de sesión para aplicaciones web que implementan técnicas estándar de administración de sesiones. A continuación, se incluyen algunas defensas a tener en cuenta al utilizar cookies de navegador.

- Cuando las cookies del navegador se utilizan como mecanismo para rastrear la sesión de un usuario autenticado, estas deben ser accesibles para un conjunto mínimo de dominios y rutas y deben etiquetarse para caducar en, o poco después, el período de validez de la sesión.
- La bandera "segura" debe establecerse para garantizar que la transferencia se realice únicamente a través de un canal seguro (TLS).
- Se debe establecer el indicador HttpOnly para evitar que se acceda a la cookie a través de JavaScript.
- Agregar atributos de "samesite" a las cookies evita que algunos navegadores modernos envíen cookies con solicitudes entre sitios y brinda protección contra la falsificación de solicitudes entre sitios y los ataques de fuga de información.

Tokens

Las sesiones del lado del servidor pueden ser limitantes para algunas formas de autenticación. Los "servicios sin estado" permiten la gestión del lado del cliente de los datos de la sesión con fines de rendimiento, por lo que el servidor tiene menos carga para almacenar y verificar la sesión del usuario.

Estas aplicaciones "sin estado" generan un token de acceso de corta duración que se puede utilizar para autenticar la solicitud de un cliente sin enviar las credenciales del usuario después de la autenticación inicial.

JWT (Jason Web Tokens)

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente. Un token JWT se crea durante la autenticación y es verificado por el servidor (o servidores) antes de cualquier procesamiento.

Sin embargo, el servidor no suele guardar los JWT después de la creación inicial. Por lo general, los JWT se crean y luego se entregan a un cliente sin que el servidor los guarde de ninguna manera. La integridad del token se mantiene mediante el uso de firmas digitales para que un servidor pueda verificar posteriormente que el JWT sigue siendo válido y no ha sido manipulado desde su creación.

Este enfoque no tiene estado y es portátil en la forma en que las tecnologías de cliente y servidor pueden ser diferentes y aun así interactuar.

Presta atención

La identidad digital, la autenticación y la gestión de sesiones son temas muy importantes. Estamos rascando la superficie del tema de la identidad digital aquí. Asegúrate de que tu talento sea capaz de mantener la complejidad involucrada con la mayoría de las soluciones de identidad.

Vulnerabilidades prevenidas

- OWASP Top 10 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 M4- Insecure Authentication

Referencias

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

7. APLICAR CONTROLES DE ACCESO

El control de acceso o autorización es el proceso de otorgar o denegar solicitudes específicas de un usuario, programa o proceso. El control de acceso también implica el acto de otorgar y revocar esos privilegios.

Cabe señalar que la autorización (verificar el acceso a recursos o funciones específicos) no es equivalente a la autenticación (verificar la identidad).

La funcionalidad de control de acceso a menudo abarca muchas áreas de software según la complejidad del sistema de control de acceso. Por ejemplo, la administración de metadatos de control de acceso o el almacenamiento en caché para propósitos de escalabilidad son a menudo componentes adicionales en un sistema de control de acceso que necesitan ser construidos o administrados.

Hay varios tipos diferentes de diseño de control de accesos que deben ser considerados:

- El control de acceso discrecional (DAC) es un medio para restringir el acceso a objetos (por ejemplo, ficheros, entidades de datos) en función de la identidad y la necesidad de conocer de los sujetos (por ejemplo, usuarios, procesos) y/o grupos a los que pertenece el objeto.
- El control de acceso obligatorio (MAC) es un medio para restringir el acceso a los recursos del sistema en función de la sensibilidad (representada por una etiqueta) de la información contenida en el recurso del sistema y la autorización formal (por ejemplo, acreditación) de los usuarios para acceder a la información de tal sensibilidad.
- El control de acceso basado en roles (RBAC) es un modelo para controlar el acceso a los recursos donde las acciones permitidas sobre los recursos se identifican con roles en lugar de identidades de sujetos individuales.

- El control de acceso basado en atributos (ABAC) otorgará o denegará las solicitudes de los usuarios en función de los atributos arbitrarios del usuario y los atributos arbitrarios del objeto, y las condiciones del entorno que serán reconocidas globalmente y más relevantes para las políticas en cuestión.

Principios de diseño de control de acceso

Los siguientes requisitos de diseño de control de acceso "positivos" deben considerarse en las etapas iniciales del desarrollo de la aplicación.

1. **Diseño de control de acceso desde el principio**

   Una vez que haya elegido un patrón de diseño de control de acceso específico, a menudo es difícil y requiere mucho tiempo rediseñar el control de acceso en su aplicación con un nuevo patrón.

   El control de acceso es una de las áreas principales del diseño de seguridad de aplicaciones que debe diseñarse minuciosamente desde el principio, especialmente cuando se abordan requisitos como el control de acceso multi cliente y horizontal (dependiente de los datos).

   El diseño del control de acceso puede comenzar simple, pero a menudo puede convertirse en un control de seguridad complejo y con muchas funciones.

   Al evaluar la capacidad de control de acceso de los frameworks de software, asegúrate de que su funcionalidad de control de acceso permita la personalización para tus necesidades específicas de función de control de acceso.

2. **Obliga a que todas las solicitudes pasen por controles de control de acceso**

Asegúrese de que todas las solicitudes pasen por algún tipo de capa de verificación de control de acceso. Las tecnologías como los filtros de Java u otros mecanismos de procesamiento automático de solicitudes son artefactos de programación ideales que ayudarán a garantizar que todas las solicitudes pasen por algún tipo de verificación de control de acceso.

3. **Denegar por defecto**

Denegar por defecto es el principio de que, si una solicitud no se permite específicamente, se rechaza. Hay muchas formas en que esta regla se manifestará en el código de la aplicación.

Algunos ejemplos de estos son:

- El código de la aplicación puede generar un error o una excepción al procesar las solicitudes de control de acceso. En estos casos, siempre se debe negar el control de acceso.
- Cuando un administrador crea un nuevo usuario o un usuario se registra para una nueva cuenta, esa cuenta debe tener un acceso mínimo o nulo de forma predeterminada hasta que se configure ese acceso.
- Cuando se agrega una nueva función a una aplicación, a todos los usuarios se les debe negar el uso de esa función hasta que esté configurada correctamente.

4. **Principio de privilegio mínimo**

Asegúrese de que a todos los usuarios, programas o procesos se les dé el mínimo o mínimo acceso necesario posible. Tenga cuidado con los sistemas que no proporcionan capacidades de configuración de control de acceso granular

5. No “Hardcodear” roles

Muchos frameworks de aplicaciones tienen de forma predeterminada un control de acceso basado en roles. Es común encontrar un código de aplicación que esté lleno de controles de esta naturaleza.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
      deleteAccount();
}
```

Tenga cuidado con este tipo de programación basada en roles en el código. Tiene las siguientes limitaciones o peligros.

- La programación basada en roles de esta naturaleza es frágil. Es fácil crear comprobaciones de roles incorrectas o faltantes en el código.
- La programación basada en roles no permite multiusuario. Se requerirán medidas extremas como bifurcar el código o verificaciones adicionales para cada cliente para permitir que los sistemas basados en roles tengan diferentes reglas para diferentes clientes.
- La programación basada en roles no permite reglas de control de acceso horizontales o específicas de datos.
- Las bases de código grandes con muchas comprobaciones de control de acceso pueden ser difíciles de auditar o verificar la política general de control de acceso de la aplicación.

En su lugar, considere la siguiente metodología de programación de control de acceso:

```
if (user.hasAccess("DELETE_ACCOUNT")) {
      deleteAccount();
}
```

Las comprobaciones de control de acceso basadas en atributos o características de esta naturaleza son el punto de partida para construir sistemas de control de acceso bien diseñados y ricos en funciones.

Este tipo de programación también permite una mayor capacidad de personalización del control de acceso a lo largo del tiempo.

6. Registrar todos los eventos de control de acceso

Todas las fallas de control de acceso deben registrarse, ya que pueden indicar que un usuario malintencionado está investigando la aplicación en busca de vulnerabilidades.

Vulnerabilidades prevenidas

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2016-M4 Insecure Authentication

Referencias

- OWASP Cheat Sheet: Access Control

Herramientas

- OWASP ZAP with the optional Access Control Testing add-on

8. PROTEJA LOS DATOS EN TODAS PARTES

Los datos sensibles tales como contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos empresariales requieren protección adicional, particularmente si esos datos están sujetos a las leyes de privacidad (Reglamento General de Protección de Datos de la UE, GDPR), reglas de protección de datos financieros como el Estándar de Seguridad de Datos de PCI (PCI DSS) u otras regulaciones.

Los atacantes pueden robar datos de la web y las aplicaciones de servicios web de varias formas. Por ejemplo, si se envía información confidencial a través de Internet sin seguridad en las comunicaciones, entonces un atacante en una conexión inalámbrica compartida podría ver y robar los datos de otro usuario. Además, un atacante podría usar SQL Injection para robar contraseñas y otras credenciales de una base de datos de aplicaciones y exponer esa información al público.

Clasificación de datos

Es fundamental clasificar los datos en su sistema y determinar a qué nivel de sensibilidad pertenece cada dato. Luego, cada categoría de datos se puede asignar a las reglas de protección necesarias para cada nivel de sensibilidad. Por ejemplo, la información de marketing público que no es confidencial puede clasificarse como datos públicos que se pueden colocar en el sitio web público. Los números de tarjetas de crédito pueden clasificarse como datos de usuario privados que pueden necesitar encriptarse mientras están almacenados o en tránsito.

Encriptación de datos en tránsito

Al transmitir datos confidenciales a través de cualquier red, se debe considerar la seguridad de las comunicaciones de extremo a extremo (o cifrado en tránsito) de algún tipo. TLS es, con mucho, el protocolo criptográfico más común y ampliamente admitido para la seguridad de las comunicaciones.

Es utilizado por muchos tipos de aplicaciones (web, servicio web, móvil) para comunicarse a través de una red de forma segura. TLS debe configurarse correctamente en una variedad de formas para defender adecuadamente las comunicaciones seguras.

El principal beneficio de la seguridad de la capa de transporte es la protección de los datos de las aplicaciones web contra la divulgación y modificación no autorizadas cuando se transmiten entre clientes (navegadores web) y el servidor de aplicaciones web, y entre el servidor de aplicaciones web y el back-end y otros componentes empresariales no basados en navegador.

Encriptar datos en reposo

La primera regla de la gestión de datos confidenciales es evitar almacenar datos confidenciales cuando sea posible. Si debe almacenar datos confidenciales, asegúrese de que estén protegidos criptográficamente de alguna manera para evitar la divulgación y modificación no autorizadas.

La criptografía es uno de los temas más avanzados de seguridad de la información y uno cuya comprensión requiere la mayor educación y experiencia.

Es difícil hacerlo bien porque hay muchos enfoques para el cifrado, cada uno con ventajas y desventajas que los arquitectos y desarrolladores de soluciones web deben comprender a fondo.

Además, la investigación seria en criptografía se basa típicamente en matemáticas avanzadas y teoría de números, lo que constituye una seria barrera de entrada.

En lugar de crear una capacidad criptográfica desde cero, se recomienda encarecidamente que se utilicen soluciones abiertas y revisadas por pares, como el proyecto Google Tink, Libsodium y la capacidad de almacenamiento seguro integrada en muchos frameworks de software y servicios en la nube.

Aplicación móvil: almacenamiento local seguro

Las aplicaciones móviles corren un riesgo particular de fuga de datos porque los dispositivos móviles se pierden o son robados con regularidad, pero contienen datos confidenciales.

Como regla general, solo se deben almacenar los datos mínimos requeridos en el dispositivo móvil. Pero si debe almacenar datos confidenciales en un dispositivo móvil, entonces los datos confidenciales deben almacenarse dentro del directorio de almacenamiento de datos específico de cada sistema operativo móvil. En Android, este será el almacén de claves de Android y en iOS será el llavero de iOS.

El ciclo de vida de las claves

Las claves secretas se utilizan en varias aplicaciones de funciones sensibles. Por ejemplo, las claves secretas se pueden utilizar para firmar JWT (JSON Web Tokens), proteger tarjetas de crédito, proporcionar diversas formas de autenticación y facilitar otras funciones de seguridad sensibles. Al administrar las claves, se deben seguir una serie de reglas que incluyen:

- Asegúrese de que cualquier clave secreta esté protegida contra el acceso no autorizado.
- Almacene las claves en una “bóveda de secretos” adecuada como se describe a continuación.
- Utilice claves independientes cuando se requieran varias claves.
- Desarrolle soporte para cambiar algoritmos y claves cuando sea necesario.
- Cree funciones de la aplicación para gestionar una rotación de claves.

Gestión de secretos de aplicaciones

Las aplicaciones contienen numerosos *"secretos"* necesarios para las operaciones de seguridad. Estos incluyen certificados, contraseñas de conexión SQL, credenciales de cuentas de servicios de terceros, contraseñas, claves SSH, claves de cifrado y más.

La divulgación o modificación no autorizada de estos secretos podría llevar a un compromiso total del sistema. Al administrar los secretos de la aplicación, tenga en cuenta lo siguiente.

- No almacene secretos en código, archivos de configuración ni los pase a través de variables de entorno. Utilice herramientas como GitRob o TruffleHog para escanear repositorios de código en busca de secretos.
- Guarde las claves y sus otros secretos a nivel de aplicación en una bóveda de secretos como KeyWhiz o el proyecto Vault de Hashicorp o Amazon KMS para proporcionar almacenamiento seguro y acceso a secretos a nivel de aplicación en tiempo de ejecución.

Vulnerabilidades prevenidas

- OWASP Top 10 2017 - A3: Datos sensibles expuestos
- OWASP Mobile Top 10 2016-M2 Almacenamiento inseguro de datos

Referencias

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Testing Guide: Testing for TLS

Herramientas

- SSLyze - SSL configuration scanning library and CLI tool
- SSLLabs - Free service for scanning and checking TLS/SSL configuration
- OWASP O-Saft TLS Tool - TLS connection testing tool
- GitRob - Command line tool to find sensitive information in publicly available files on GitHub
- TruffleHog - Searches for secrets accidentally committed
- KeyWhiz - Secrets manager
- Hashicorp Vault - Secrets manager
- Amazon KMS - Manage keys on Amazon AWS

9. IMPLEMENTAR SEGURIDAD EN LOGGING Y MONITORIZACIÓN

El logging es un concepto que la mayoría de los desarrolladores ya utilizan con fines de depuración y diagnóstico. Securizar el logging es un concepto igualmente básico: para dar seguridad a la información de log durante las operaciones de ejecución de una aplicación.

El monitoreo es la revisión en vivo de la aplicación y la seguridad de los logs usando varias formas de automatización. Las mismas herramientas y patrones se pueden utilizar para operaciones, depuración y seguridad.

Beneficios de seguridad en el logging

Securizar el logging se puede usar para:

- Como fuente de información y partida para sistemas de detección de intrusos
- Análisis e investigaciones forenses
- Satisfacer los requisitos de cumplimiento normativo

Implementación de seguridad en el logging

Lista de las mejores prácticas de implementación para la seguridad del logging:

- Seguir un formato y enfoque de logging común dentro del sistema y en todos los sistemas de una organización. Un ejemplo de un framework de logging es "Apache Logging Services", que ayuda a proporcionar consistencia de logging entre las aplicaciones Java, PHP, .NET y C ++.
- No registre demasiado o muy poco. Por ejemplo, asegúrate de registrar siempre la marca de tiempo y la información de identificación, incluida la IP de origen y la identificación de usuario, pero tenga cuidado de no registrar datos privados o confidenciales.

- Preste mucha atención a la sincronización de tiempo entre nodos para asegurarse de que las marcas de tiempo sean coherentes.

Logging para la detección y respuesta de intrusiones

Utilice el logging para identificar la actividad que indica que un usuario se está comportando de forma maliciosa.

La actividad potencialmente maliciosa para registrar incluye:

- Datos enviados que están fuera de un rango numérico esperado.
- Datos enviados que involucran cambios a datos que no deben ser modificables (lista de selección, casilla de verificación u otro componente de entrada limitada).
- Solicitudes que violan las reglas de control de acceso del lado del servidor.

Cuando la aplicación encuentra tal actividad, la aplicación debe al menos registrar la actividad y marcarla como un problema de gravedad alta. Idealmente, la aplicación también debería responder a un posible ataque identificado, por ejemplo, invalidando la sesión del usuario y bloqueando la cuenta del usuario.

Los mecanismos de respuesta permiten al software reaccionar en tiempo real ante posibles ataques identificados.

Diseño seguro de logging

Un diseño seguro de logging debe incluir lo siguiente:

- Codifique y valide cualquier carácter peligroso antes de iniciar sesión para evitar ataques de inyección en el registro o de falsificación de registros (log Injection / log forging).
- No registre información confidencial. Por ejemplo, no registre contraseña, ID de sesión, tarjetas de crédito o números de seguro social.

- Proteja la integridad del registro. Un atacante puede intentar alterar los registros. Por lo tanto, se debe considerar el permiso de los archivos de registro y la auditoría de cambios de registro.
- Reenviar registros de sistemas distribuidos a un servicio de registro central y seguro. Esto asegurará que los datos de registro no se pierdan si un nodo está comprometido. Esto también permite un monitoreo centralizado.

Referencias

- OWASP Log injection

Herramientas

Apache Logging Services

10. MANEJAR TODOS LOS ERRORES Y EXCEPCIONES

El manejo de excepciones es un concepto de programación que permite que una aplicación responda a diferentes estados de error (como red inactiva, conexión de base de datos fallida, etc.) de varias formas. Manejar las excepciones y los errores correctamente es fundamental para que el código sea confiable y seguro.

El manejo de errores y excepciones se produce en todas las áreas de una aplicación, incluida la lógica de negocio crítica, así como las funciones de seguridad y el código del framework.

El manejo de errores también es importante desde la perspectiva de la detección de intrusos. Ciertos ataques contra su aplicación pueden desencadenar errores que pueden ayudar a detectar ataques en curso.

Error en el manejo de errores

Los investigadores de la Universidad de Toronto han descubierto que incluso los pequeños errores en el manejo de errores o el olvido de manejar los errores pueden conducir a fallos catastróficos en los sistemas distribuidos.

Los errores en el manejo de errores pueden generar diferentes tipos de vulnerabilidades de seguridad:

- Fuga de información: la filtración de información confidencial en mensajes de error puede ayudar involuntariamente a los atacantes. Por ejemplo, un error que devuelve un seguimiento de pila u otros detalles de error interno puede proporcionar a una atacante información sobre su entorno. Incluso las pequeñas diferencias en el manejo de diferentes condiciones de error (por ejemplo, devolver "usuario no válido" o "contraseña no válida" en los errores de autenticación) pueden proporcionar pistas valiosas para los atacantes. Como se describió anteriormente, asegúrese de registrar los detalles del error para fines forenses y de depuración, pero no exponga esta información, especialmente a un cliente externo.

- Omisión de TLS: El "error de fallo" de Apple goto fue un error del control de flujo en el código de manejo de errores que llevó a un compromiso completo de las conexiones TLS en los sistemas Apple.
- DOS: La falta de manejo básico de errores puede provocar el apagado del sistema. Esta suele ser una vulnerabilidad bastante fácil de explotar para los atacantes. Otros problemas de manejo de errores podrían conducir a un mayor uso de la CPU o el disco de formas que podrían degradar el sistema.

Consejos

- Administre las excepciones de manera centralizada para evitar bloques try / catch duplicados en el código. Asegúrese de que todo comportamiento inesperado se maneje correctamente dentro de la aplicación.
- Asegúrese de que los mensajes de error que se muestran a los usuarios no filtran datos críticos, pero siguen siendo lo suficientemente detallados para permitir la respuesta adecuada del usuario.
- Asegúrese de que las excepciones se registren de manera que brinden suficiente información para que los equipos de soporte, control de calidad, forenses o de respuesta a incidentes comprendan el problema.
- Pruebe y verifique cuidadosamente el código de manejo de errores.

Referencias

- OWASP Code Review Guide: Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

Herramientas

- Error Prone: una herramienta de análisis estático de Google para detectar errores comunes en el manejo de errores para los desarrolladores de Java
- Una de las herramientas automatizadas más famosas para encontrar errores en tiempo de ejecución es Chaos Monkey de Netflix, que deshabilita intencionalmente las instancias del sistema para garantizar que el servicio general se recupere correctamente.

11. PUNTOS CLAVE

Una vez vistos todos los controles proactivos que podemos aplicar en nuestros desarrollos estamos preparados para poder llevar a cabo un desarrollo mucho más seguro en nuestras aplicaciones a nivel de:

> Desarrollo

> Testing

> Auditoría